DATA 301

Python II

Dr. Firas Moosvi
University of British Columbia Okanagan
firas.moosvi@ubc.ca

Python Math Expressions

Math ***expressions*** in Python:

Operation	Syntax	Example
Add	+	5 + 3
Subtract	-	10 – 2
Multiply	*	5 * 3
Divide	/	9 / 4
Modulus	%	9 % 4 (answer is 1)
Exponent	**	5 ** 2 (answer is 25)

Expressions - Operator Precedence

Each operator has its own priority similar to their priority in regular math expressions:

- 1) Any expression in parentheses is evaluated first starting with the inner most nesting of parentheses.
- 2) Exponents
- 3) Multiplication and division (*, /, %)
- 4) Addition and subtraction (+,-)

Python Expressions Question

Question: What is the value of this expression:

```
8 ** 2 + 12 / 4 * (3 - 1) % 5
```

A) 69 **B)** 65 **C)** 36 **D)** 16 **E)** 0

Try it: Python Variables and Expressions

Question 1: Write a program that prints the result of `35 + 5 * 10`.

Question 2: Write a program that uses at least 3 operators to end up with the value `99`.

Question 3: Write a program that has a variable called `name` with the value of your name and a variable called `age` storing your age. Print out your name and age using these variables.

Strings

Strings are sequences of characters that are surrounded by either single or double quotes.

- Use \ to escape ' E.g. There\'s
- Can use triple double quotes """ for a string that spans multiple lines.

Example:

```
name = "Joe Jones"

storeName = 'Joe\'s Store'
print("""String that is really long
with multiple lines
    and spaces is perfectly fine""")
```

Python String Indexing

Individual characters of a string can be accessed using square brackets (`[]`) with the first character at index 0.

Example:

```
str = "Hello"
print(str[1])          # e
print("ABCD"[0])       # A
print(str[-1])         # o
# Negative values start at end and go backward
```

Rules for Strings in Python

Must be surrounded by single or double quotes.

Can contain most characters except enter, backspace, tab, and backslash.

- These special characters must be escaped by using an initial "`\`".
- e.g. `\n` – new line, `\'` – single quote, `\\` - backslash, `\"` – double quote
- A string in raw mode (`r` before quote) will ignore backslash escape. May be useful if data contains escapes. Example: `st = r"slash\there\"`

Double quoted strings can contain single quoted strings and vice versa.

Any number of characters is allowed.

The minimum number of characters is zero `""`, which is called the *empty string*.

String *literals* (values) have the quotation marks removed when displayed.

Python Strings Question

Question: How many of the following are valid Python strings?

1) `""`
2) `''`
3) `"a"`
4) `"  "`
5) `""""`
6) `"Joe\' Smith\""`

A) 1 **B)** 2 **C)** 3 **D)** 4 **E)** 5

Python String Functions

```
st  = "Hello"
st2 = "Goodbye"
```

Operation	Syntax	Example	Output
Length	`len()`	`len(st)`	`5`
Upper case	`upper()`	`st.upper()`	`HELLO`
Lower case	`lower()`	`st.lower()`	`hello`
Convert to a string	`str()`	`str(9)`	`"9"`
Concatenation	`+`	`st1 + st2`	`HelloGoodbye`

Substring	`[]`	`st[0:3]`
		`st[1:]`
			`Hel`
			`ello`

String to int	`int()`	`int("99")`	`99`

String Operators: Concatenation

The ***concatenation operator*** is used to combine two strings into a single string. The notation is a plus sign '**+**'.

Example:

```
st1 = "Hello"
st2 = "World!"
st3 = st1 + st2 # HelloWorld!
print(st1+st1)
num = 5
print(st1+str(num))    # Hello5
# Must convert number to string before
# concatenation
```

String Concatenation Question

Question: What is the output of this code?

```
st1 = "Hello"
st2 = "World!"
num = 5
print(st1 + str(num) + " " + st2)
```

A) `Error`

B) `Hello5World!`

C) `Hello5 World!`

D) `Hello 5 World!`

Substring

The ***substring*** function will return a range of characters from a string.

Syntax:

```
st[start:end]   # start is included, end is not
                # first character is index 0
```

Examples:

```
st = "Fantastic"
print(st[1])          # a
print(st[0:6])        # Fantas
print(st[4:])         # astic
print(st[:5])         # Fanta
print(st[-6:-2])      # tast
```

Substring Question

Question: What is the output of this code?

```
st = "ABCDEFG"
print(st[1] + st[2:4] + st[3:] + st[:4])
```

A) `ABCDCDEFGABCD`

B) `ABCDEFGABC`

C) `ACDDEFGABCD`

D) `BCDDEFGABCD`

E) `BCDECDEFGABC`

Split

The *split* function will divide a string based on a separator.

Examples:

```
st = "Awesome coding! Very good!"
print(st.split())
# ['Awesome', 'coding!', 'Very', 'good!']
print(st.split("!"))
# ['Awesome coding', ' Very good', '']
st = 'data,csv,100,50,,25,"use split",99'
print(st.split(","))
# ['data', 'csv', '100', '50', '', '25',
#   '"use split"', '99']
```

Try it: Python String Variables and Functions

Question 1: Write a Python program that prints out your name and age stored in variables like this:

```
Name: Joe
Age: 25
```

Question 2: Write a Python program that prints out the first name and last name of `Steve Smith` like below. You must use substring.

- Bonus challenge: Use `find()` function so that it would work with any name.

```
First Name: Steve
Last Name: Smith
```

Print Formatting

The `print` method can accept parameters for formatting.

```
print("Hi", "Amy", ", your age is", 21)
print("Hi {}, your age is {}".format("Amy",21))
```

This is one of the most obvious changes between Python 2:

```
print "Hello"
```

and Python 3:

```
print("Hello")
```

Python Date and Time

Python supports date and time data types and functions.

First, import the `datetime` module:

```
from datetime import datetime
```

Functions:

```
now = datetime.now()
print(now)
current_year = now.year
current_month = now.month
current_day = now.day
print("{}-{}-{} {}:{}:{}".format(now.year, now.month,
now.day, now.hour, now.minute, now.second))
```

Python Clock

Python `time()` function returns the current time in seconds:

```
import time
startTime = time.time()
print("Start time:", startTime)
print("How long will this take?")
endTime = time.time()
print("End time:", endTime)
print("Time elapsed:", endTime-startTime)
```

Python Input

To read from the keyboard (standard input), use the method `input`:

```
name = input("What's your name?")
print(name)
age = input("What's your age?")
print(age)
```

Prompt for value from user

print out value received

- Note in Python 2 the method is called `raw_input()`.

Try it: Python Input, Output, and Dates

Question 1: Write a program that reads a name and prints out the name, the length of the name, the first five characters of the name.

Question 2: Print out the current date in `YYYY/MM/DD` format.

Comparisons

A ***comparison operator*** compares two values. Examples:

- `5 < 10`
- `N > 5   # N is a variable. Answer depends on what is N.`

Comparison operators in Python:

`>`	- Greater than
`>=`	- Greater than or equal
`<`	- Less than
`<=`	- Less than or equal
`==`	- Equal (Note: Not "=" which is used for assignment!)
`!=`	- Not equal

The result of a comparison is a ***Boolean value*** which is either **`True`** or **`False`**.

Conditions with `and`, `or`, `not`

A ***condition*** is an expression that is either `True` or `False` and may contain one or more comparisons. Conditions may be combined using: `and`, `or`, `not`.

- order of evaluation: `not`, `and`, `or` May change order with parentheses.

Operation	Syntax	Examples	Output

AND
(True if both are True)
	`and`	`True and True`
		`False and True`
		`False and False`
			`True`
			`False`
			`False`

OR
(True if either or both are True)
	`or`	`True or True`
		`False or True`
		`False or False`
			`True`
			`True`
			`False`

NOT
(Reverses: e.g. True becomes False)
	`not`	`not True`
		`not False`
			`False`
			`True`

Condition Examples

```
n = 5
v = 8
print(n > 5)                            # False
print(n == v)                           # False
print(n != v)                           # True
print(n == v and n+4>v)                 # False
print(n == v or n+4>v)                  # True
print(n+1 == v-2 or not v>4)            # True
```

Python Condition Question

Question: How many of the following conditions are **TRUE**?

1) `True and False`
2) `not True or not False`
3) `3 > 5 or 5 > 3 and 4 != 4`
4) `(1 < 2 or 3 > 5) and (2 == 2 and 4 != 5)`
5) `not (True or False) or True and (not False)`

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Decisions

Decisions allow the program to perform different actions based on conditions. Python decision syntax:

```
if condition:
    statement
```

Done if condition is `True`

```
if condition:
    statement
else:
    statement
```

Done if condition is `False`

- The statement after the `if` condition is only performed if the condition is `True`.
- If there is an `else`, the statement after the else is done if condition is `False`.
- Indentation is important! Remember the colon!

Decisions `if/elif` Syntax

If there are more than two choices, use the `if/elif/else` syntax:

```
if condition:
    statement
elif condition:
    statement
elif condition:
    statement
else:
    statement
```

```
if n == 1:
    print("one")
elif n == 2:
    print("two")
elif n == 3:
    print("three")
else:
    print("Too big!")
print("Done!")
```

Decisions: Block Syntax

Statements executed after a decision in an `if` statement are indented for readability. This indentation is also how Python knows which statements are part of the block of statements to be executed.

- If you have more than one statement, make sure to indent them. Be consistent with either using tabs or spaces. Do not mix them!

```
if age > 19 and name > "N":
    print("Not a teenager")
    print("Name larger than N")
else:
    print("This is statement #1")
    print(" and here is statement #2!")
```

Question: Decisions

Question: What is the output of the following code?

```
n = 3
if n < 1:
    print("one")
elif n > 2:
    print("two")
elif n == 3:
    print("three")
```

A) nothing **B)** `one` **C)** `two` **D)** `three`

Question: Decisions (2)

Question: What is the output of the following code?

```
n = 3
if n < 1:
    print("one")
elif n > 2
    print("two")
else:
    print("three")
```

A) nothing

B) `one`

C) `two`

D) `three`

E) error

Question: Decisions (3)

Question: What is the output of the following code?

```
n = 1
if n < 1:
    print("one")
elif n > 2:
    print("two")
else:
    print("three")
print("four")
```

A) nothing

B) `one`
`four`

C) `three`

D) `three`
`four`

E) error

Question: Decisions (4)

Question: What is the output of the following code?

```
n = 0
if n < 1:
    print("one")
    print("five")
elif n == 0:
    print("zero")
else:
    print("three")
print("four")
```

A) nothing

B)
```
one
four
```

C)
```
one
five
four
```

D)
```
one
five
zero
four
```

E) error

Try it: Decisions

Question 1: Write a Python program that asks the user for a number then prints out if it is even or odd.

Question 2: Write a Python program that asks the user for a number between 1 and 5 and prints out the word for that number (e.g. 1 is one). If the number is not in that range, print out error.

DATA 301
Extra Reference slides about Python

Dr. Firas Moosvi
University of British Columbia Okanagan
firas.moosvi@ubc.ca

Loops and Iteration

A ***loop*** repeats a set of statements multiple times until some condition is satisfied.

- Each time a loop is executed is called an ***iteration***.

A `for` loop repeats statements a number of times.

A `while` loop repeats statements while a condition is `True`.

The `while` Loop

The most basic looping structure is the ***while*** loop.

A while loop continually executes a set of statements **while** a condition is true.

Syntax:

```
while condition:
    statements
```

Example:

```
n = 1
while n <= 5:
    print(n)
    n = n + 1      # Shorthand: n += 1
```

Question: What does this print?

Question: `while` Loop

Question: What is the output of the following code?

```
n = 4
while n >= 0:
    n = n - 1
    print(n)
```

A) numbers 3 to -1 **B)** numbers 3 to 0 **C)** numbers 4 to 0

D) numbers 4 to -1 **E)** numbers 4 to infinity

Question: `while` Loop (2)

Question: What is the output of the following code?

```
n = 1
while n <= 5:
    print(n)
n = n + 1
```

A) nothing **B)** numbers 1 to 5 **C)** numbers 1 to 6 **D)** lots of 1*s*

The `for` Loop

A `for` loop repeats statements a given number of times.

Python `for` loop syntax:

Up to but not including ending number

```
for i in range(1,6):
    print(i)
```

Starting number

Using `range`

The basic form of range is:

```
range(start,end)
```

- start is inclusive, end is not inclusive
- default increment is 1

May also specify an increment:

```
range(start, end, increment)
```

or just the end:

```
range(end)
```

For Loop and While Loop

The `for` loop is like a short-hand for the `while` loop:

```
i=0
while i < 10:
    print(i)
    i += 1
```

```
for i in range(0, 10, 1):
    print(i)
```

Common Problems – Infinite Loops

Infinite loops are caused by an incorrect loop condition or not updating values within the loop so that the loop condition will eventually be false.

Example:

```
n = 1
while n <= 5:
    print(n)
    # Forgot to increase n -> infinite loop
```

Common Problems – Off-by-one Error

The most common error is to be "***off-by-one***". This occurs when you stop the loop one iteration too early or too late.

Example:

- This loop was supposed to print 0 to 10, but it does not.

```
for i in range(0,10):
    print(i)
```

Question: How can we fix this code to print 0 to 10?

Question: `for` Loop

Question: How many numbers are printed with this loop?

```
for i in range(1,10):
   print(i)
```

A) 0 **B)** 9 **C)** 10 **D)** 11 **E)** error

Question: `for` Loop

Question: How many numbers are printed with this loop?

```
for i in range(11,0):
    print(i)
```

A) 0 **B)** 9 **C)** 10 **D)** 11 **E)** error

Try it: `for` Loops

Question 1: Write a program that prints the numbers from 1 to 10 then 10 to 1.

Question 2: Write a program that prints the numbers from 1 to 100 that are divisible by 3 and 5.

Question 3: Write a program that asks the user for 5 numbers and prints the maximum, sum, and average of the numbers.

Lists Overview

A ***list*** is a collection of data items that are referenced by index.

- Lists in Python are similar to arrays in other programming languages

A list allows multiple data items to be referenced by one name and retrieved by index.

Python list:

```
data = [100, 200, 300, 'one', 'two', 600]
         0    1    2     3      4     5
```

list variable name

Indexes

Retrieving Items from a List

Items are retrieved by index (starting from 0) using square brackets:

```
data = [100, 200, 300, 'one', 'two', 600]
print(data[0])                # 100
print(data[4])                # 'two'
print(data[6])                # error – out of range
print(data[len(data)-1])      # 600
print(data[-1])                 # 600
print(data[2:4])              # [300, 'one']

# Create an empty list:
emptyList = []
```

List Operations

```
data = [1, 2, 3, 5]
lst = []
```

Operation	Syntax	Examples	Output
Add item	*list*.append(*val*)	data.append(1)	[1, 2, 3, 5, 1]
Insert item	*list*.insert(*idx*,*val*)	data.insert(3,4)	[1, 2, 3, 4, 5]
Remove item	*list*.remove(*val*)	data.remove(5)	[1, 2, 3]
Update item	*list*[*idx*]=*val*	lst[0]=10	[10]
Length of list	len(*list*)	len(data)	4
Slice of list	*list*[x:y]	data[0:3]	[1, 2, 3]
Find index	*list*.index(*val*)	data.index(5)	3
Sort list	*list*.sort()	data.sort()	[1, 2, 3, 5]

List Details

If you provide an index outside of the valid range, Python will return an index error.

To sort in reverse order, do this:

```
data.sort(reverse=True)
```

For loops are used to iterate though items in a list:

```
for v in data:
   print(v)
```

Advanced: Python Lists Comprehensions

List comprehensions build a list using values that satisfy a criteria.

```
evenNums100 = [n for n in range(101) if n%2==0]
```

Equivalent to:

```
evenNums100 = []
for n in range(101):
    if n%2==0:
        evenNums100.append(n)
```

Advanced: Python Lists Slicing

List slicing allows for using range notation to retrieve only certain elements in the list by index. Syntax:

```
list[start:end:stride]
```

Example:

```
data = list(range(1,11))
print(data)       # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(data[1:8:2])  # [2, 4, 6, 8]
print(data[1::3])   # [2, 5, 8]
```

Question: List

Question: At what index is item with value 3?

```
data = [1, 2, 3, 4, 5]
data.remove(3)
data.insert(1, 3)
data.append(2)
data.sort()
data = data[1:4]
```

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** not there

Try it: Lists

Question 1: Write a program that puts the numbers from 1 to 10 in a list then prints them by traversing the list.

Question 2: Write a program that will multiply all elements in a list by 2.

Question 3: Write a program that reads in a sentence from the user and splits the sentence into words using split(). Print only the words that are more than 3 characters long. At the end print the total number of words.

Python Dictionary

A ***dictionary*** is a collection of key-value pairs that are manipulated using the key.

```
dict = {1:'one', 2:'two', 3:'three'}
print(dict[1])              # one
print(dict['one'])          # error - key not found
if 2 in dict:               # Use in to test for key
   print(dict[2])           # two
dict[4] = 'four'            # Add 4:'four'
del dict[1]                 # Remove key 1
dict.keys()                 # Returns keys
dict.values()               # Returns values
```

Question: Dictionary

Question: What is the value printed?

```
data = {'one':1, 'two':2, 'three':3}
data['four'] = 4
sum = 0
for k in data.keys():
   if len(k) > 3:
      sum = sum + data[k]
print(sum)
```

A) 7 **B)** 0 **C)** 10 **D)** 6 **E)** error

Try it: Dictionary

Question: Write a program that will use a dictionary to record the frequency of each letter in a sentence. Read a sentence from the user then print out the number of each letter.

Code to create the dictionary of letters:

```
import string
counts = {}
for letter in string.ascii_uppercase:
    counts[letter] = 0
print(counts)
```

Functions and Procedures

A ***procedure*** (or ***method***) is a sequence of program statements that have a specific task that they perform.

- The statements in the procedure are mostly independent of other statements in the program.

A ***function*** is a procedure that returns a value after it is executed.

We use functions so that we do not have to type the same code over and over. We can also use functions that are built-in to the language or written by others.

★ Defining and Calling Functions and Procedures

Creating a function involves writing the statements and providing a ***function declaration*** with:

- a name (follows the same rules as identifiers)
- list of the inputs (called parameters)
- the output (return value) if any

Calling (or executing) a function involves:

- providing the name of the function
- providing the values for all arguments (inputs) if any
- providing space (variable name) to store the output (if any)

Defining and Calling a Function

Consider a function that returns a number doubled:

def Keyword

Function Name

Parameter Name

```
def doubleNum(num):
    num = num * 2
    print("Num: "+num)
    return num
```

Call function by name

Argument

```
n = doubleNum(5)                  # 10
print(str(doubleNum(n)))          # ??
```

Python Built-in Math Functions

```
# Math
import math
print(math.sqrt(25))

# Import only a function
from math import sqrt
print(sqrt(25))

# Print all math functions
print(dir(math))
```

Other Python Built-in Functions

`max`, `min`, `abs`:

```
print(max(3, 5, 2))     # 5
print(min(3, 5, 2))     # 2
print(abs(-4))          # 4
```

`type()` returns the argument data type:

```
print(type(42))           # <class 'int'>
print(type(4.2))        # <class 'float'>
print(type('spam'))     # <class 'str'>
```

Python Random Numbers

Use random numbers to make the program have different behavior when it runs.

```
from random import randint
coin = randint(0, 1)              # 0 or 1
die = randint(1, 6)              # 1 to 6
print(coin)
print(die)
```

Advanced: Python Functions

Python supports functional programming allowing functions to be passed like variables to other functions.

- Lambda functions are functions that do not have a name.

Example:

```
def doFunc(func, val):
    return func(val)

print(doFunc(doubleNum, 10))          # 20
print(doFunc(lambda x: x * 3, 5))     # 15
```

Question: Functions

Question: What is the value printed?

```
def triple(num):
   return num * 3

n = 5
print(triple(n)+triple(2))
```

A) 0 **B)** 6 **C)** 15 **D)** 21 **E)** error

Practice Questions: Functions

1) Write a function that returns the largest of two numbers.

2) Write a function that prints the numbers from 1 to N where N is its input parameter.

Call your functions several times to test that they work.

Conclusion

Python is a general, high-level programming language designed for code readability and simplicity.

Programming concepts covered:

- variables, assignment, expressions, strings, string functions
- making decisions with conditions and `if/elif/else`
- repeating statements (loops) using `for` and `while` loops
- reading input with `input()` and printing with `print()`
- data structures including lists and dictionaries
- creating and calling functions, using built-in functions (math, random)

Python is a powerful tool for data analysis and automation.

Objectives

- Explain what is Python and note the difference between Python 2 and 3
- Define: algorithm, program, language, programming
- Follow Python basic syntax rules including indentation
- Define and use variables and assignment
- Apply Python variable naming rules
- Perform math expressions and understand operator precedence
- Use strings, character indexing, string functions
- String functions: split, substr, concatenation
- Use Python datetime and clock functions
- Read input from standard input (keyboard)

Objectives (2)

- Create comparisons and use them for decisions with `if`
- Combine conditions with `and`, `or`, `not`
- Use `if`/`elif`/`else` syntax
- Looping with `for` and `while`
- Create and use lists and list functions
- Advanced: list comprehensions, list slicing
- Create and use dictionaries
- Create and use Python functions
- Use built-in functions in math library
- Create random numbers
- Advanced: passing functions, lambda functions